

# Protetto di recupero (Progetto B) del modulo di laboratorio SOL 2016

## Contents

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Materiale in linea . . . . .	1
1.2	Struttura del progetto e tempi di consegna . . . . .	1
1.3	Valutazione del progetto . . . . .	2
<b>2</b>	<b>memBox++</b>	<b>3</b>
2.1	Descrizione . . . . .	3
2.2	Esecuzione e file di configurazione . . . . .	4
2.3	File di dump . . . . .	5
2.4	Gestione dei segnali . . . . .	6
2.5	Statistiche . . . . .	6
2.6	Protocollo client server . . . . .	6
2.7	Script bash . . . . .	6
<b>3</b>	<b>Istruzioni</b>	<b>7</b>
3.1	Materiale fornito dai docenti . . . . .	7
3.2	Cosa devono fare gli studenti . . . . .	8
<b>4</b>	<b>Parti Opzionali</b>	<b>8</b>
<b>5</b>	<b>Codice e documentazione</b>	<b>8</b>
5.1	Vincoli sul codice . . . . .	8
5.2	Formato del codice . . . . .	9
5.3	Relazione . . . . .	9

## 1 Introduzione

Il modulo di Laboratorio di Programmazione di Sistema del corso di Sistemi Operativi e Laboratorio (277AA) prevede lo svolgimento di un progetto da sviluppare utilizzando C in ambiente Linux. Il progetto può essere svolto singolarmente dallo studente o al più da due studenti appartenenti allo stesso corso (a meno di differente accordo con i docenti). Questo documento descrive la struttura complessiva del progetto di recupero (chiamato anche Progetto B) che dovrà essere sviluppato dagli studenti che non hanno consegnato il Progetto A entro la scadenza del 30/06/2016. La scadenza per la consegna del Progetto B coincide con la scadenza dell'appello straordinario (Marzo/Aprile 2017)

dell'a.a 2016. Il progetto dovrà essere sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentati durante il corso.

## 1.1 Materiale in linea

Tutto il materiale relativo al corso può essere reperito sul sito web:

<http://didawiki.cli.di.unipi.it/doku.php/informatica/sol/laboratorio16>

Il sito verrà progressivamente aggiornato con le informazioni riguardanti il progetto (es. FAQ, suggerimenti, avvisi, update delle specifiche), il ricevimento e simili. Il sito è un Wiki e gli studenti possono registrarsi per ricevere automaticamente gli aggiornamenti delle pagine che interessano maggiormente. In particolare consigliamo a tutti di registrarsi alla pagina delle 'FAQ' e degli 'Avvisi Urgenti'.

Eventuali chiarimenti possono essere richiesti consultando i docenti del corso durante l'orario di ricevimento, le ore in laboratorio e/o per posta elettronica.

## 1.2 Struttura del progetto e tempi di consegna

Il progetto di recupero denominato **memBox++** è una estensione del progetto **membox** proposto nel Progetto A (frammento 2). **memBox++** prevede delle parti aggiuntive rispetto a quanto proposto nel Progetto A, in particolare:

- devono essere implementate le nuove operazioni **LOCK\_OBJ\_OP**, **UNLOCK\_OBJ\_OP** e **DUMP\_OP** (see Sec. 2)
- deve essere gestita l'opzione fornita a linea di comando **-d dumpfile**

Il progetto può essere consegnato entro l'appello straordinario dell'a.a 2016. La consegna del progetto avviene *esclusivamente* per posta elettronica. Deve essere generato un tarball utilizzando il target **consegna** del **Makefile** contenuto nel kit di sviluppo fornito dai docenti (il nome del kit è **kit.f2b.tar**). Il tarball generato deve essere allegato ad un messaggio e-mail con oggetto:

**lso16: Progetto B - corso A (oppure corso B)**

ed inviato all'indirizzo di posta elettronica

**lso.di@listgateway.unipi.it**

Non è necessario (e non deve essere fatto) iscriversi alla lista per inviare il messaggio con il progetto. Le consegne sono seguite da un messaggio di conferma (non automatico) da parte di uno dei docenti del corso. La notifica verrà inviata all'indirizzo di posta da cui è pervenuta la consegna del progetto. Se la ricezione non viene confermata entro 3/4 giorni lavorativi, contattare il docente per e-mail.

Si prega di specificare nella mail che allega il progetto, Nome e Cognome dello studente (o degli studenti) che hanno sviluppato il progetto **e corso di appartenenza – Corso A o Corso B**.

*I progetti che non contengono tutti i file necessari o che non compilano ed eseguono correttamente su versioni recenti di Ubuntu x86\_64 non saranno accettati. I progetti che non rispettano il formato o il cui tarball non è stato generato con il target **consegna** non verranno accettati. Si prega di controllare che tutti i file necessari alla corretta compilazione ed esecuzione del progetto siano presenti nel tarball prima di inviarlo.*

*La consegna del progetto deve comunque essere fatta almeno 15 giorni prima della data dell'appello che si intende sostenere. Si ricorda, che per sostenere l'esame è necessario iscriversi.*

## 1.3 Valutazione del progetto

Al progetto viene assegnata una valutazione complessiva da 0 a 30. La *qualità della documentazione allegata* da un contributo fino a 6 punti. La valutazione del progetto è effettuata in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, corretto uso del Makefile e librerie etc.)
- efficienza e robustezza del software
- modalità di testing
- aderenza alle specifiche
- gestione della concorrenza e dei segnali
- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione (vedi Sez. 5.3)

La prova orale tenderà a stabilire se lo studente è realmente l'autore del progetto consegnato e verterà su tutto il programma del corso e su tutto quanto usato nel progetto anche se non fa parte del programma del corso. L'esito della prova orale è essenziale per delineare il voto finale. In particolare, l'orale potrà comprendere:

- una discussione delle scelte implementative del progetto
- l'impostazione e la scrittura di semplici script bash e Makefile
- l'impostazione e la scrittura di programmi C + POSIX calls non banali (sia sequenziali che concorrenti)
- domande su tutto il programma presentato durante il corso.

**Casi particolari** Gli studenti lavoratori iscritti alla laurea triennale in Informatica possono consegnare il progetto ottenendo tutti i bonus in qualsiasi appello dell'AA 2015/16. In questo caso è necessaria la certificazione da consegnare al docente come da regolamento di ateneo.

Gli studenti che svolgono il progetto per le lauree di secondo livello sono invitati a contattare il docente.

Gli studenti iscritti ai vecchi ordinamenti (nei quali il voto di Lab. 4 contribuiva al voto di SO) avranno assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

## 2 memBox++

### 2.1 Descrizione

Lo scopo del progetto è lo sviluppo di un server concorrente che implementa uno spazio di memorizzazione di *oggetti (repository)*. Un oggetto è una sequenza continua di bytes di dimensione non nulla ed avente una size massima stabilita nel file di configurazione del server con l'opzione **MaxObjSize** (vedi Sez. 2.2). L'oggetto è identificato univocamente da una chiave numerica. Ogni richiesta inviata al server deve contenere la chiave dell'oggetto a cui la richiesta si riferisce.

Uno o più processi client comunicano con il server per inviare richieste di vario di tipo. In particolare le richieste possono essere: di inserimento (PUT\_OP), di aggiornamento di un oggetto già inserito (UPDATE\_OP), di rimozione di un oggetto (REMOVE\_OP) presente nel repository, di recupero (GET\_OP) del contenuto di un oggetto, di acquisizione della lock (LOCK\_OP) sull'intero repository e rilascio della stessa (UNLOCK\_OP), di acquisizione della lock su un singolo oggetto (LOCK\_OBJ\_OP)

e rilascio della stessa (UNLOCK\_OBJ\_OP), di dump del repository (DUMP\_OP) sul disco.

Ad esempio, una richiesta di recupero di un oggetto con chiave 100 sarà codificata inviando un messaggio di richiesta al server contenente le informazioni <100, GET\_OP> opportunamente formattate. Il server risponderà con un messaggio di risposta contenente nella parte dati i bytes dell'oggetto con chiave 100. Se l'oggetto non è presente nel repository (ad esempio perché è stato rimosso da un altro processo), verrà ritornato al client un messaggio di errore (OP.GET\_NONE – vedere il file `ops.h`).

La comunicazione avviene tramite socket di tipo AF\_UNIX. Il path da utilizzare per la creazione del socket AF\_UNIX deve essere specificato nel file di configurazione utilizzando l'opzione `UnixPath`. Per rendere più agevole lo sviluppo ed il testing dell'applicazione sulle macchine del centro di calcolo per la didattica, le socket create dallo studente con numero di matricola `matricola` potranno avere un nome che rispetta il seguente formato: `membox_sock_matricola`. L'obiettivo è minimizzare possibili interazioni indesiderate fra progetti sviluppati da utenti diversi sulla stessa macchina del laboratorio.

Le operazioni di LOCK\_OP ed UNLOCK\_OP si intendono sull'intero repository e non sul singolo oggetto (per il singolo oggetto l'operazione è LOCK\_OBJ\_OP). Se il server è in stato "locked" dovrà servire solamente le richieste provenienti dal client che mantiene la lock. Se il client chiude la connessione, la lock sull'intero repository dovrà essere rilasciata anche se non è stata inviata una esplicita operazione di unlock.

A seguito di una operazioni di LOCK\_OBJ\_OP, se l'oggetto è già in stato di lock allora il client che ha fatto la richiesta deve attendere fino a quando non verrà rilasciata la lock. La gestione della lock sull'oggetto deve essere gestita in modo FIFO e deve essere evitato deadlock se lo stesso client effettua due volte l'operazione di LOCK\_OBJ\_OP sullo stesso oggetto.

L'operazione di dump (DUMP\_OP) può essere eseguita solo se prima il client ha acquisito la lock sull'intero repository. Il file che dovrà contenere il dump di tutti gli oggetti presenti nel repository è indicato nel file di configurazione con l'opzione `DumpFile`. Se tale opzione non è presente, al client deve essere ritornato un opportuno messaggio di errore. Ad ogni richiesta di dump deve essere generato un nuovo file. Il vecchio file, se presente, deve essere rinominato appendendo al suo nome il suffisso `.bak`. Se esiste già un file con lo stesso nome ed estensione `.bak`, quest'ultimo verrà sovrascritto.

Il server può contenere un numero massimo di oggetti pari a `StorageSize` e non dovrà occupare una size complessiva per gli oggetti maggiore di `StorageByteSize` (se `StorageSize` e/o `StorageByteSize` valgono 0 vuol dire che non c'è nessun limite su numero massimo di oggetti/dimensione dello storage). Oltre tali valori di soglia il server non dovrà più accettare richieste di inserimento di nuovi oggetti, mentre dovrà continuare ad accettare altri tipi di richieste (rimozione, aggiornamento, recupero, ...). Nello sviluppo del codice del server si dovrà tenere in considerazione che:

- il server deve essere in grado di gestire decine di migliaia di oggetti
- il server deve essere in grado di gestire efficientemente alcune decine di connessioni contemporanee.

Il server al suo interno implementa un sistema multi-threaded che è in grado di gestire contemporaneamente sia nuove connessioni dai client che gestire le richieste su connessioni già stabilite. Un cliente può sia inviare una richiesta per singola connessione oppure, per una data connessione, inviare più richieste (sia sullo stesso oggetto che su oggetti diversi). Il numero massimo di connessioni concorrenti che il server `memBox++` gestisce è stabilito dall'opzione di configurazione `MaxConnections`. Se tale numero viene superato, dovrà essere ritornato al client un opportuno messaggio di errore. Ogni connessione viene gestita interamente da un solo thread (se lo studente lo ritiene, quest'ultimo requirement può essere rilassato, ossia una connessione può

essere gestita anche da più threads). Il numero di thread utilizzati per gestire le connessioni è specificato nel file di configurazione con l'opzione **ThreadsInPool**. Tale numero specifica la dimensione di un pool di threads, sempre attivi per tutta la durata dell'esecuzione del server, che il server **memBox++** utilizzerà per gestire le connessioni. Tale valore non può essere 0. Qualora tutti i thread del pool risultino occupati, possono essere accodate fino a  $MaxConnections - ThreadsInPool$  connessioni in attesa che il pool le serva. Un thread appartenente al pool, non appena conclude la gestione di una connessione, controlla se ci sono connessioni pendenti in coda ed in caso affermativo ne prende una da gestire. Se non ci sono connessioni in coda, il thread "rientra" nel pool di thread disponibili in attesa di essere svegliato per gestire una nuova connessione.

## 2.2 Esecuzione e file di configurazione

Il server **memBox++** viene attivato da shell con il comando

```
$ membox -f membox.conf [-d dumpfile]
```

dove **membox.conf** è il file di configurazione contenente tutti i parametri relativi alla configurazione del server. Il formato del file di configurazione è schematizzato in Fig. 1,

```
# path utilizzato per la creazione del socket AF_UNIX
UnixPath          = /tmp/mbox_socket

# numero massimo di connessioni pendenti
MaxConnections    = 32

# numero di thread nel pool
ThreadsInPool     = 8

# dimensione dello storage in numero di oggetti (0 nessun limite)
StorageSize       = 0
# dimensione dello storage in numero di bytes (0 nessun limite)
StorageByteSize   = 0

# dimensione massima di un oggetto nello storage (0 nessun limite)
MaxObjSize        = 0

# file nel quale verranno scritte le statistiche
StatFileName      = /tmp/mbox_stats.txt

# file nel qual verra' effettuato il dump del repository
DumpFile = /tmp/mbox.dump
```

Figure 1: Formato del file testuale **membox.conf**

L'argomento opzionale **dumpfile** indica al server di caricare tutti gli oggetti presenti nel file di dump (generato da una esecuzione precedente). Se tale opzione è presente, il contenuto del file **dumpfile** deve essere caricato nel repository. Il formato del file di dump è riportato nella Sez. 2.3. Va notato che il file passato con l'opzione **-d** ed il file specificato con l'opzione **DumpFile** nel file di configurazione, non sono necessariamente lo stesso file.

Il processo `memBox++` termina quando viene ricevuto il segnale `SIGINT` o `SIGTERM` o `SIGQUIT`. Alla ricezione di tale segnale il server esce il prima possibile dopo che ha segnalato a tutti i thread attivi di chiudere immediatamente tutte le connessioni e dopo aver stampato le statistiche nel file specificato nel file di configurazione con l'opzione `StatFileName`. Tutti i socket unix ed eventuali file temporanei devono essere cancellati all'uscita del server.

## 2.3 File di dump

Il formato del file contenente il dump di tutti gli oggetti presenti nel repository è riportato in Fig. 2

```
numero di oggetti
size dello storage
object1-key
object1-size
object1-data
object2-key
object2-size
object2-data
....
```

Figure 2: Formato del file binario `mbox.dump`

## 2.4 Gestione dei segnali

Oltre ai segnali `SIGINT` o `SIGTERM` o `SIGQUIT` che determinano la terminazione “immediata” di `memBox++`, il server gestisce altri 2 segnali: `SIGUSR1` e `SIGUSR2`. Se viene inviato un segnale di tipo `SIGUSR1` il server stampa le statistiche correnti nel file associato all'opzione `StatFileName` presente nel file di configurazione secondo il formato descritto nella sezione seguente. Se tale opzione è vuota o non è presente, il segnale ricevuto viene ignorato. Se viene inviato un segnale di tipo `SIGUSR2` il server smette di accettare nuove connessioni in ingresso e, prima di terminare, aspettare che tutte le connessioni in corso vengano chiuse senza forzare l'uscita immediata di tutti i thread.

## 2.5 Statistiche

Alla ricezione del segnale `SIGUSR1` il server appende nel file `StatFileName` le statistiche del server. Le stampa nel file deve avere il formato `<timestamp - statdata>` dove `timestamp` è l'istante temporale in cui le statistiche vengono inserite nel file e `statdata` è una sequenza di numeri (separati da uno spazio) che riportano i valori monitorati durante il tempo di vita del server. I dati statistici minimi che devono essere prodotti nel file sono riportati in Fig. 3. Se lo studente lo ritiene, può arricchire le statistiche prodotte documentandole opportunamente nella relazione che accompagna il progetto.

## 2.6 Protocollo client server

Il protocollo di comunicazione per ogni tipo di richiesta che il client può inviare al server deve essere stabilito dallo studente e documentato nella relazione. Per effettuare i test deve essere utilizzato il client (`client.c`) fornito dai docenti. In particolare, devono essere definiti chiaramente i tipi dei messaggi di richiesta e risposta e come sono gestiti

```

n. di PUT_OP ricevute      n. di PUT_OP fallite
n. di UPDATE_OP ricevute   n. di UPDATE_OP fallite
n. di GET_OP ricevute      n. di GET_OP fallite
n. di REMOVE_OP ricevute   n. di REMOVE_OP fallite
n. di LOCK_OP ricevute     n. di LOCK_OP fallite
n. di LOCK_OBJ_OP ricevute n. di LOCK_OBJ_OP fallite
n. di DUMP_OP ricevute     n. di DUMP_OP fallite
n. di connessioni concorrenti
n. massimo di oggetti memorizzati raggiunto
n. corrente di oggetti memorizzati
size massima in bytes raggiunta
size in bytes corrente

```

Figure 3: Dati statistici minimi che devono essere prodotti nel file `mbox_stats.txt`

eventuali messaggi di errore che il server ritorna al client (ad esempio a seguito di una operazione di tipo GET\_OP di un oggetto non esistente nel repository).

## 2.7 Script bash

Deve essere realizzato un script bash che dato in input il file delle statistiche produca in output opportune informazione sulla base dei parametri passati allo script.

Lo script si attiva da linea di comando con diverse opzioni

```

mboxstat.sh statfile
mboxstat.sh statfile -p -u -g -r -c -s -o -m -d
mboxstat.sh --help

```

Senza nessuna opzione lo script stampa tutte le statistiche dell'ultimo timestamp. L'opzione `-p` stampa il numero di PUT\_OP ed il numero di PUT\_OP fallite per tutti i timestamp. L'opzione `-u` stampa il numero di UPDATE\_OP ed il numero di UPDATE\_OP fallite per tutti i timestamp. L'opzione `-g` stampa il numero di GET\_OP ed il numero di GET\_OP fallite per tutti i timestamp. L'opzione `-r` stampa il numero di REMOVE\_OP ed il numero di REMOVE\_OP fallite per tutti i timestamp. L'opzione `-c` stampa il numero di connessioni per tutti i timestamp. L'opzione `-s` stampa la size in KB per tutti i timestamp. L'opzione `-o` stampa il numero di oggetti nel repository per tutti i timestamp. L'opzione `-m` stampa il massimo numero di connessioni raggiunte dal server, il massimo numero di oggetti memorizzati ed la massima size in KB raggiunta. L'opzione `-d` stampa il numero di dump per tutti i timestamp. Infine l'opzione lunga `help` stampa un messaggio di uso dello script.

Le opzioni possono apparire in un qualsiasi ordine sulla linea di comando e più opzioni possono essere usate contemporaneamente.

È fatto esplicito divieto di usare `sed` o `awk` nella realizzazione dello script.

Come esempio di output dello script, si consideri il seguente caso:

```

> cat /tmp/mbox_stats.txt

...
1468081427 - 16509 5 422 2 2165 14 16768 283 29 0 0 0 0 3 8220419 18706179 24 1045
1468081428 - 17533 5 423 2 2167 14 16768 283 29 0 0 0 0 3 8875267 18706179 1048 1048
1468081429 - 17534 5 425 2 2167 14 17794 283 29 0 0 0 0 3 6195300 18706179 23 1048
1468081429 - 17598 5 425 2 2231 14 17858 283 29 0 0 0 0 3 6195300 18706179 23 1048

```

```
1468081433 - 17600 5 427 2 2234 14 17864 286 30 0 0 0 0 0 6184189 18706179 22 1048
...
```

```
> ./memboxstat.sh -p -r /tmp/mbox_stats.txt
TIMESTAMP      PUT      PUTF      REMOVE      REMOVEF
.....
1468081434      17681      5        17956      301
1468081435      18706      5        17958      302
1468081436      18707      5        18986      303
1468081436      18771      5        19050      303
1468081440      18774      5        19058      306
1468081441      18774      5        19058      306
.....
```

## 3 Istruzioni

### 3.1 Materiale fornito dai docenti

Viene fornito un kit denominato `kit_f2b.tar` contenente i file per lo svolgimento del progetto `memBox++`. Il kit può essere scaricato all'indirizzo:

<http://didawiki.cli.di.unipi.it/doku.php/informatica/sol/laboratorio16/progetto>

Si prega di leggere con attenzione il `README` contenuto nel kit per una descrizione dei file forniti dai docenti.

### 3.2 Cosa devono fare gli studenti

Gli studenti devono:

- leggere *attentamente* le specifiche del progetto contenute in questo documento, il `README` contenuti nel kit e capire il funzionamento del codice fornito dai docenti (in particolare il `client.c`)
- implementare tutte le funzionalità richieste
- verificare le funzioni con i test forniti dai docenti (attenzione: questi test vanno eseguiti su codice già corretto e funzionante altrimenti possono dare risultati fuorvianti o di difficile interpretazione)
- preparare la *documentazione*: vedi Sez. 5.
- generare i progetti finali esclusivamente utilizzando il `Makefile` fornito dai docenti seguendo le istruzioni riportate nei `README`.
- creare un unico tarball contenente i tarball dei due progetti ed inviarlo ai docenti seguendo le istruzioni fornite nella Sez. 1.2.

## 4 Parti Opzionali

Possono essere realizzate funzionalità ed opzioni in più rispetto a quelle richieste (ad esempio per quanto riguarda `memBox++`, aggiungendo altre opzioni nel file di configurazione aumentando le funzionalità del server, inserendo nuove statistiche, fornendo grafici per le statistiche etc.).

Le parti opzionali devono essere spiegate nella relazione e corredate di test appropriati ove opportuno.



## 5 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

### 5.1 Vincoli sul codice

*Makefile e codice devono compilare ed eseguire CORRETTAMENTE su versioni recenti di Ubuntu x86\_64 o su un insieme non vuoto delle macchine del Centro di Calcolo (CDC). La relazione deve specificare su quali versioni di Ubuntu o su quali macchine CDC è possibile far girare correttamente il codice.*

La stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire attraverso il Makefile fornito dai docenti
- il codice deve compilare senza errori o warning gravi utilizzando le opzioni `-Wall -pedantic`
- NON devono essere utilizzate funzioni di temporizzazione quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi/thread. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi/thread coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)
- il server non deve generare memory leaks (testare l'esecuzione del server con `valgrind`)

### 5.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne (se non ovvio dal contesto)

### 5.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 15 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi. In particolare, NON devono essere ripetute le specifiche contenute in questo documento.* In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e loro motivazioni)
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- la struttura dei programmi sviluppati
- la struttura dei programmi di test aggiuntivi oltre a quelli forniti dai docenti
- l'interazione fra i processi/threads ed i relativi protocolli (es. terminazione, gestione concorrenza, segnali)
- le difficoltà incontrate e le soluzioni adottate
- eventuali estensioni che lo studente ha sviluppato oltre a quanto richiesto
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- README di istruzioni su come compilare/eseguire ed utilizzare il codice

La relazione deve essere in formato PDF.